

CST8130: Data Structures: Linked Lists Non-Nested versus Nested Classes

Adding at the Head with Only a Head Pointer

```
// Non-Nested Implementation of Linked List Classes
#include <iostream>
using namespace std;

class List; // forward declaration because of friend declaration

class ListNode { // Non-Nested but Related Class Specification: The Dependent
friend class List;
private:
    char* pszName;
    ListNode* pInNext;
public:
    ListNode() : pszName(NULL), pInNext(NULL) { }
    ListNode(char* pszData) : pszName(NULL), pInNext(NULL)
    {
        if (pszData != NULL)
            pszName = strcpy(new char[strlen(pszData) + 1], pszData);
    }
    ~ListNode() { if (pszName != NULL) delete [ ] pszName; }
}; // end Non-Nested but Related Class Specification: Dependent

class List { // Non-Nested but Related Class Specification: The Manager
private:
    int nNumItems;
    ListNode* pInHead;
public:
    List() : nNumItems(0), pInHead(NULL) { }
    virtual ~List() { DeleteAll(); } // virtual not needed now, but supports inheritance
    void DeleteAll();
    void AddToHead(char* pszNewData);
    void OutputAllRecords(ostream& ros);
}; // end Non-Nested but Related Class Specification: The Manager
```

```
void List::AddToHead(char* pszNewData)
{
    ListNode* pInTmp = new ListNode(pszNewData);
    pInTmp->pInNext = pInHead;
    pInHead = pInTmp;
    nNumItems++;
}

void List::DeleteAll()
{
    while (pInHead != NULL) {
        ListNode* pInToDelete = pInHead;
        pInHead = pInHead->pInNext;
        delete pInToDelete;
    }
}

void List::OutputAllRecords(ostream& ros)
{
    ListNode* pInCurrent = pInHead;
    while (pInCurrent != NULL) {
        ros << pInCurrent->pszName << endl;
        pInCurrent = pInCurrent->pInNext;
    }
}

void main()
{
    List lPeople;
    char cMore;
    do {
        char szNewName[128];
        cout << "Enter a name:";
        cin >> szNewName;
        lPeople.AddToHead(szNewName);
        cout << "More? (y/n):";
        cin >> cMore;
    } while (cMore == 'y');

    cout << "\nList of People\n\n";
    lPeople.OutputAllRecords(cout);
}
```

Nested Class Implementation

```
// Nested Implementation of Linked List Classes
#include <iostream>
using namespace std;

class List { // Nested Class Specification: The Manager
private:
    class ListNode { // Nested Class Specification: The Dependent
public:
    char* pszName;
    ListNode* pInNext;
    ListNode() : pszName(NULL), pInNext(NULL) { }
    ListNode(char* pszData) : pszName(NULL), pInNext(NULL)
    {
        if (pszData != NULL)
            pszName = strcpy(new char[strlen(pszData) + 1], pszData);
    }
    ~ListNode() { if (pszName != NULL) delete [ ] pszName; }
}; // end ListNode Nested Class Specification: The Dependent

    int nNumItems;
    ListNode* pInHead;
public:
    List() : nNumItems(0), pInHead(NULL) { }
    virtual ~List() { DeleteAll(); } // virtual not needed now, but supports inheritance
    void DeleteAll();
    void AddToHead(char* pszNewData);
    void OutputAllRecords(ostream& ros);
}; // end Nested Class Specification: The Manager

void List::AddToHead(char* pszNewData)
{
    ListNode* pInTmp = new ListNode(pszNewData);
    pInTmp->pInNext = pInHead;
    pInHead = pInTmp;
    nNumItems++;
}
}
```

```
void List::DeleteAll()
{
    while (pInHead != NULL) {
        ListNode* pInToDelete = pInHead;
        pInHead = pInHead->pInNext;
        delete pInToDelete;
    }
}

void List::OutputAllRecords(ostream& ros)
{
    ListNode* pInCurrent = pInHead;
    while (pInCurrent != NULL) {
        ros << pInCurrent->pszName << endl;
        pInCurrent = pInCurrent->pInNext;
    }
}

void main()
{
    List IPeople;
    char cMore;
    do {
        char szNewName[128];
        cout << "Enter a name:";
        cin >> szNewName;
        IPeople.AddToHead(szNewName);
        cout << "More? (y/n):";
        cin >> cMore;
    } while (cMore == 'y');

    cout << "\nList of People\n\n";
    IPeople.OutputAllRecords(cout);
}
}
```